developer WORLD

Developers guidelines

June 2010

3D graphics with Java™ Platform, Micro Edition

for Sony Ericsson phones

Sony Ericsson

Preface

Purpose of this document

This document describes 3D support for Java™ Platform, Micro Edtion (Java ME™), found in Sony Ericsson phones. The document also describes how to develop Java 3D applications based on the Mascot Capsule® Micro3D Version 3 and the Mobile 3D Graphics API for J2ME (JSR-184).

Readers who will benefit from this document include:

- Software developers
- Corporate buyers
- IT professionals.
- Support engineers
- Business decision makers

It is assumed that the reader is familiar with Java.

These Developers guidelines are published by:

Sony Ericsson Mobile Communications AB,
SE-221 88 Lund, Sweden

www.sonyericsson.com/

© Sony Ericsson Mobile Communications AB, 2004. All rights reserved. You are hereby granted a license to download and/or print a copy of this document.
Any rights not expressly granted herein are reserved.

39th version (June 2010)
Publication number: EN/LZT 198 7387 R39A

This document is published by Sony Ericsson Mobile Communications AB, without any warranty*. Improvements and changes to this text necessitated by typographical errors, inaccuracies of current information or improvements to programs and/or equipment, may be made by Sony Ericsson Mobile Communications AB at any time and without notice. Such changes will, however, be incorporated into new editions of this document. Printed versions are to be regarded as temporary reference copies only.

*All implied warranties, including without limitation the implied warranties of merchantability or fitness for a particular purpose, are excluded. In no event shall Sony Ericsson or its licensors be liable for incidental or consequential damages of any nature, including but not limited to lost profits or commercial loss, arising out of the use of the information in this document.

Sony Ericsson Developer World

At www.sonyericsson.com/developer, developers find the latest technical documentation and development tools such as phone White papers, Developers guidelines for different technologies, Getting started tutorials, SDKs (Software Development Kits) and tool plugins. The Web site also features news articles, go-to-market advice, moderated discussion forums offering free technical support and a Wiki community sharing expertise and code examples.

For more information about these professional services, go to the Sony Ericsson Developer World Web site.

Document conventions

Products

Sony Ericsson phones are referred to in this document by generic names (for information about Sony Ericsson Java platforms, JP-3, JP-4 and so on, Entry Java platform EJP-1.0, Sony Ericsson Java platform for Windows® Mobile® and Sony Ericsson Java platform for S60, see “Sony Ericsson Java platforms” on page 12):

Generic names Series	**Sony Ericsson phones**
No Sony Ericsson Java platform:	
S302	S302
W302	W302
Sony Ericsson Java platform for Windows Mobile:	
X2	Xperia™ X2, Xperia™ X2a
Aspen	Sony Ericsson Aspen™ M1i, Sony Ericsson Aspen™ M1a
Sony Ericsson Symbian™ Java platform:	
Satio™	Satio™ U1i, Satio™ U1a, Satio™ U1
Vivaz	Sony Ericsson Vivaz™ U5i, Sony Ericsson Vivaz™ U5a
Vivaz pro	Sony Ericsson Vivaz™ pro U8i, Sony Ericsson Vivaz™ pro U8a
EJP-1.0 phones:	
F305	F305, F305c
JP-3 phones:	
F500	F500i

Generic names Series	**Sony Ericsson phones**
J300	J300i, J300c, J300a
K300	K300i, K300c, K300a
K500	K500i, K506c, K508i, K508c
K700	K700i, K700c
S700	S700i, S700c, S710a
Z500	Z500a
JP-4 phones:	
V800	V800, Vodafone 802SE
Z800	Z800i
JP-5 phones:	
K600	K600i, K608i
K750	K750i, K750c, D750i
V600	V600i
W700	W700i, W700c
W800	W800i, W800c
Z520	Z520i, Z520c, Z520a
Z525	Z525a
JP-6 phones:	
K310	K310i, K310c, K310a
K320	K320i, K320c
K510	K510i, K510c
W200	W200i, W200c
W300	W300i, W300c
W550	W550i, W550c
W600	W600i
W810	W810i, W810c, W810a
W900	W900i
Z530	Z530i, Z530c
Z550	Z550i, Z550c, Z550a
Z558	Z558i, Z558c

Generic names Series	**Sony Ericsson phones**
JP-7 phones:	
K530	K530i
K550	K550i, K550c
K610	K610i, K610c, K618i
K770	K770i
K790	K790i, K790c, K790a
K800	K800i, K800c
K810	K810i, K818c
S500	S500i, S500c
T650	T650i, T658c
W350	W350i, W350c
W380	W380i, W380c
W580	W580i, W580c
W610	W610i, W610c
W660	W660i
W710	W710i, W710c
W830	W830i, W830c
W850	W850i, W850c
W880	W880i, W888c
Z555	Z555i, Z555c
Z610	Z610i
Z710	Z710i, Z710c

Generic names Series	**Sony Ericsson phones**
JP-8 (8.0-8.2) phones:	
G502	G502, G502c
K630	K630i
K660	K660i
K850	K850i, K858c
V640	V640i
W890	W890i
W910	W910i, W908c
Z750	Z750i
Z770	Z770i
JP-8.3 phones:	
C702	C702, C702c, C702a
C902	C902, C902c
T700	T700
W595	W595, W595s
W760	W760i, W760c
W902	W902
W980	W980i
Z780	Z780i, Z780a

Generic names Series	**Sony Ericsson phones**
JP-8.4 phones:	
C510	C510, C510c, C510a
C901	C901, C901a, C901 GreenHeart™
C903	C903, C903a
C905	C905, C905c, C905a
G705	G705, G705u
Jalou™	Jalou™ F100i, BeJoo™ F100i
Naite™	Naite™ J105i, Naite™ J105a
T707	T707, T707a
T715	T715, T715a
W508	W508, W508c, W508a, W518a
W705	W705, W705u
W715	W715
W995	W995, W995a
JP-8.5 phones:	
Aino™	Aino™ U10i, Aino™ U10a
Cedar	Sony Ericsson Cedar™ J108i, Sony Ericsson Cedar™ J108a
Elm	Sony Ericsson Elm™ J10, Sony Ericsson Elm™ J10i2
Hazel	Sony Ericsson Hazel™ J20, Sony Ericsson Hazel™ J20i
Pureness™	Xperia™ Pureness™ X5, Xperia™ Pureness™ X5i
Yari™	Yari™ U100i, Yari™ U100a
Zylo	Sony Ericsson Zylo™ W20, Sony Ericsson Zylo™ W20i

Typographical conventions

In this document, code is written in Courier font, for example:

```
Vector3D position = new Vector3D( 0, 100, 256 );
```

Trademarks and acknowledgements

GreenHeart, Aino, BeJoo, Jalou, Naite, Pureness, Satio, Sony Ericsson Aspen, Sony Ericsson Cedar, Sony Ericsson Elm, Sony Ericsson Hazel, Sony Ericsson Zylo, Sony Ericsson Vivaz, Xperia and Yari are trademarks or registered trademarks of Sony Ericsson Mobile Communications AB.

Windows and Windows Mobile are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Symbian is a trademark or a registered trademark of Symbian Software Ltd.

Sun, Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the U.S. and other countries.

Mascot Capsule is a trademark of HI Corporation.

OpenGL is a registered trademark of Silicon Graphics, Inc. (SGI).

Bluetooth™ is a trademark or registered trademark of Bluetooth SIG Inc.

Other product and company names mentioned herein may be the trademarks of their respective owners.

Document history

Change history		
2004-07-04	Version R1A	First version
2004-08-23	Version R2A	Second version. Minor editorial changes. K500 series and Z500 series information added
2004-09-20	Version R2B	Second revised version. Minor editorial changes
2004-10-19	Version R3A	Third version. Editorial changes. V800 series added
2004-10-28	Version R3B	Minor editorial changes
2005-03-09	Version R4A	Information about K300, J300, K750, W800, Z800 and K600 series added
2005-04-13	Version R4B	Minor editorial changes
2005-08-01	Version R5A	Information about V600, S600, W600 and Z520 series added

2005-08-08	Version R5B	S600 series phones changed to W550
2005-09-30	Version R6A	Sixth version
2005-10-24	Version R7A	Seventh version. Information about W900 series added
2005-10-27	Version R7B	Minor editorial changes
2006-01-04	Version R8A	Eighth version. Information about W810 series added
2006-02-13	Version R9A	Ninth version. Information about K610 series added
2006-02-28	Version R10A	Tenth version. Information about K800, K790, Z530, W300, K510 and K310 series added
2006-04-12	Version R11A	11th version. Information about W700 and Z525 series added
2006-05-19	Version R12A	12th version. Information about Z550, W850, Z710 and W710 series added
2006-08-22	Version R13A	13th version. Information about the K618i and the Z610 series added
2006-09-25	Version R14A	14th version. Information about K320, Z558 and W830 series and the Z550a phone model added
2007-01-08	Version R15A	15th version. Information about W200 series added
2007-02-06	Version R16A	16th version. Information about W880, K550, W610 and K810 series added
2007-03-13	Version R17A	17th version. Information about W660 series added
2007-03-27	Version R18A	18th version. Information about W580 and Z750 series added
2007-07-02	Version R19A	19th version. Information about K530, K850, S500, T650 and W910 series added
2007-08-21	Version R20A	20th version. Information about K770 series added
2007-11-06	Version R21A	21st version. Information about K630, K660, V640, W380 and W890 series added
2008-04-04	Version R22A	22nd version. Information about C702, C902, W350, W760, W980, Z555 and Z770 series added
2008-05-20	Version R23A	23rd version. Information about G502 and Z780 series added
2008-07-22	Version R24A	24th version. Information about C905, S302, T700, W302, W595 and W902 series added

2008-09-09	Version R25A	25th version. Information about G705 series added
2008-10-17	Version R25B	25th revised version. New document layout
2008-11-11	Version R26A	26th version. Information about W705 series added
2009-01-09	Version R27A	27th version. Information about C510 and W508 series, and Entry Java Platform EJP-1.0 added
2009-02-15	Version R28A	28th version. Information about C901, C903, W715 and W995 series added
2009-03-26	Version R29A	29th version. Information about T707 series added
2009-05-28	Version R30A	30th version. Information about Aino™ and Yari™ series added
2009-06-25	Version R31A	31st version. Information about Naite™ and T715 series added
2009-08-12	Version R32A	32nd version. Information about Jalou™ series added
2009-10-26	Version R33A	33rd version. Information about Pureness™, Satio™ and X2 series added
2009-12-14	Version R34A	34th version. Information about Elm and Hazel series added
2010-01-21	Version R35A	35th version. Information about Vivaz series added
2010-02-02	Version R36A	36th version. Information about Aspen series added
2010-02-14	Version R37A	37th version. Information about Vivaz pro series added
2010-04-13	Version R38A	38th version. Information about Zylo series added
2010-06-16	Version R39A	39th version. Information about Cedar series added

Contents

Overview **12**
Sony Ericsson Java platforms 12
The 3D engines 17
Java ME integration 17
Micro3D Ver.3 features 18
JSR-184 – Mobile 3D Graphics API for the Java ME platform 19
JSR-239 – Java binding for the OpenGL ES API 19
Sony Ericsson SDK for the Java™ ME platform 20

Mascot Capsule Micro3D Version 3 ..... **22**
Graphics commands 22
Object import 22
Micro3D Ver.3 classes 23

Mobile 3D Graphics API for the Java ME platform (JSR 184) **25**
Graphics commands 25
Object import 25
JSR-184 classes 26

Getting started **29**
Micro3D Version 3 model conversion 29
Conversion flow for Micro3D Ver.3 30
Java ME platform application set up for Micro3D Ver.3 31
Mobile 3D Graphics API (JSR-184) 33

References **35**
Useful links 35
Sony Ericsson 3D Knowledge Base 35
General 35
Mascot Capsule Version 3 36
Mascot Capsule Version 4 (JSR-184) 40

Overview

There is a growing need for mobile devices to process and display 3D content. Such 3D content might be part of a game, such as its scenery and characters. Or it might be a program that uses 3D graphics to display information in a unique way.

Sony Ericsson phones offer built-in 3D graphics support to handle such content. This section provides a brief overview of the 3D technology in Sony Ericsson phones. It is assumed that the reader has a general knowledge about 3D graphics and MIDP.

Sony Ericsson Java platforms

Sony Ericsson uses a platform approach to Java implementation allowing developers to focus on a platform rather than on a variety of different product names. Four platform branches exist, supporting Symbian™ phones (SJP), Windows® Mobile® phones, feature phones (JP) and Entry phones (EJP) respectively. The platforms are implemented through an evolutionary approach in order to ensure forwards compatibility between platform versions. Normally each platform version is used in several phone models.

A list of Sony Ericsson Java platform versions for the phones in this document can be found below. Some platform features are optional, that is, configurable. For example, the Bluetooth™ APIs (JSR-82) are only enabled for phones who actually support Bluetooth wireless technology.

JP = Sony Ericsson Java platform

Note: All platforms are backwards compatible, which means that all JSRs (except the optional) implemented on one platform are also implemented on all higher platforms.

Java Platform Phones	Features	Optional features (JSR-82, JSR-256, VSCL 2.0) and comments
JP-8.5 Aino™, Cedar Elm, Hazel, Pureness™, Yari™, Zylo	• **Additional JSR-211 content handlers** • **JSR-256 and JSR-238 extensions** • **Sony Ericsson Network bearer API** • **OpenGL ES 1.1 extension pack**	**JSR-82**: All JP-8.5 phones **JSR-256**: All JP-8.5 phones
JP-8.4 C510, C901, C903, C905, G705, Jalou™, Naite™, T707, T715, W508, W705, W715, W995	• **Additional JSR-211 content handlers** • **Project Capuchin API** • **JSR-256 extensions** • **Sony Ericsson multimedia services API**	**JSR-82**: All JP-8.4 phones **JSR-256**: All JP-8.4 phones with sensor(s)

Java Platform Phones	Features	Optional features (JSR-82, JSR-256, VSCL 2.0) and comments
JP-8.3 C702, C902, T700, W595, W760, W902, W980, Z780	• **Additional JSR-211 content handlers** • **PIMChangeListener API** • **JSR-256 extensions** • **Improved security handling**	**JSR-82**: All JP-8.3 phones **JSR-256**: All JP-8.3 phones with sensor(s)
JP-8.0 - JP-8.2 G502, K630, K660, K850, V640, W890, W910, Z750, Z770	**JSR-211**	**JSR-82**: All JP-8 phones **JSR-256**: All JP-8 phones with sensor(s)
	JSR-179	
	JSR-177	
	JSR-239	
	JSR-238	
	JSR-229	
	JSR-226	
	JSR-180	
	JSR-184, version 1.1	
	JSR-248 (MSA)	
	MIDP 2.1	
JP-7 K550, K610, K770, K790, K800, K810, S500, T650, W350, W380, W580, W610, W660, W710, W830, W850, W880, Z310, Z555, Z610, Z710	**JSR-234 (Camera capabilities)**	**JSR-82: Not** W380, Z310
JP-6 K310, K320, K510, W200, W300, W550, W600, W810, W900, Z530, Z550, Z558	**JSR-205**	**JSR-82: Not** K310, W200
	JSR-172	
JP-5 K600, K750, V600, W700, W800, Z520, Z525	**JSR-75**	**JSR-82:** All JP-5 phones **VSCL 2.0**: V600 only
JP-4 V800, Z800		**VSCL 2.0**: V800 only
JP-3 F500, J300, K300, K500, K700, S700, Z500	**JSR-184 version 1.0**	
	Mascot Capsule Ver. 3	

Java Platform Phones	Features	Optional features (JSR-82, JSR-256, VSCL 2.0) and comments
JP-2 Z1010	**Nokia UI API 1.1**	
	JSR-135	
	JSR-120	
	JTWI (JSR-185)	
	MIDP 2.0	
	CLDC 1.1	

EJP = Sony Ericsson Entry Java platform.

Entry Java Platform Phones	Features	Optional features and comments
EJP-1.0 F305	**JSR-75 (PDA optional packages**	
	Mascot Capsule Ver. 3	
	JSR-184 (Mobile 3D Graphics API 1.1)	
	Nokia UI API 1.1	
	JSR-135 (MMAPI 1.2)	
	JSR-120 (WMA 1.1)	
	JSR-185 (JTWI 1.0)	
	JSR-139 (CLDC 1.1)	
	JSR-118 (MIDP 2.1)	

JP-WM_SEMC = Sony Ericsson Java platform for Windows Mobile.

Java Platform Phones	Features	Optional features and comments
JP-WM_SEMC-2.0 Aspen, X2	**JSR-75 (PDA optional packages)**	FC, ContactList, TodoList, EventList
	JSR-120 (WMA 1.1)	SMS Push
	JSR-135 (MMAPI 1.2)	
	JSR-118 (MIDP 2.1)	
	NokiaUI API 1.1	
	JSR-184 (Mobile 3D Graphics API 1.1)	Mascot Capsule V4
	JSR-185 (JTWI 1.0)	
	JSR-139 (CLDC 1.1)	
	JSR-172 (WSA 1.0)	
	JSR-177 (SATSA 1.0)	Crypto package
	JSR-179 (LAPI 1.0)	
	JSR-205 (WMA 2.0)	MMS Push
	JSR-234 (AMMS 1.0)	Image process
	JSR-238 (MIAPI 1.0)	
	JSR-256 (Sensor API 1.2)	Accelerometer **Not** supported on Aspen
	Pushregistry & Autostart	
	Capuchin	
	UIActivityMenu	
	Java Panel	
	Chunked heap support	
	Midlet Scaling support	
	JNI support	Java Native Interface
	SEMC AMS API2.0	

SJP = Sony Ericsson Symbian OS Java platform.

Java Platform Phones	Features	Comments
SJP-5.0 Satio™, Vivaz™, Vivaz™ pro	**JSR-75 (PDA optional packages)**	S60 platform specification
	JSR-82	S60 platform specification. Bluetooth and OBEX API
	JSR-120 (WMA 1.1)	S60 platform specification
	JSR-205 (WMA 2.0)	S60 platform specification
	JSR-135 (MMAPI)	S60 platform specification.
	JSR-118 (MIDP 2.1)	S60 platform specification
	Nokia UI API 1.1	S60 platform specification
	JSR-139 (CLDC 1.1)	S60 platform specification
	JSR-248 Subset (MSA)	S60 platform specification
	JSR-172 (WSA 1.0)	S60 platform specification
	JSR-177 (SATSA 1.0)	S60 platform specification. SATSA-PKI and SATSA-CRYPTO
	JSR-179 (LAPI 1.0)	S60 platform specification
	JSR-211 (CHAPI)	Sony Ericsson addition
	JSR-234 (AMMS 1.1)	Partial support, Sony Ericsson addition
	JSR-256 (Sensor API 1.2)	Sony Ericsson addition
	JSR-184 (Mobile 3D graphics API)	S60 platform specification
	JSR-239 (Java Binding for the OpenGL ES API)	Sony Ericsson addition (not supported in early versions of Satio™)
	JSR-226 (SVG API)	S60 platform specification
	eSWT	S60 platform specification
	Project Capuchin	Sony Ericsson addition
	Nokia IAPinfo API	S60 platform specification

The 3D engines

The following 3D engines are supported on the Sony Ericsson phones in this document:

- Mascot Capsule Engine Version 3, hereafter referred as "**Micro3D Ver.3**"
  Not supported on Sony Ericsson Windows Mobile Java platform and Sony Ericsson Symbian OS Java platform

- The mobile standards Mobile 3D Graphics API for the Java ME platform, hereafter referred as "**JSR-184**". On Sony Ericsson Java Platforms JP-3 to JP-7, Sony Ericsson Windows Mobile Java platform and Sony Ericsson Symbian OS Java platform, Mascot Capsule V4 is used to implement JSR-184.
  On JP-8 and higher, JSR-184 is based on a hardware accelerated Nvidia (fomerly Hybrid) implementation.

- Sony Ericsson feature phones on JP-8 platform or higher and phones on Sony Ericsson Symbian OS Java platform 5.0 or higher support JSR-239 – Java binding for the OpenGL® ES API, allowing rendering of low-level graphical objects in Java ME.

  Feature phones on JP-8.5 and higher also support the OpenGL ES 1.1 extension pack.

  Phones on Sony Ericsson Symbian OS Java platform 5.0 or higher, except early released Satio™ phones, support OpenGL ES 1.0.

The 3D engines consist of a compact software engine that renders real-time 3D graphics within memory and processing power constraints of the phone platform.

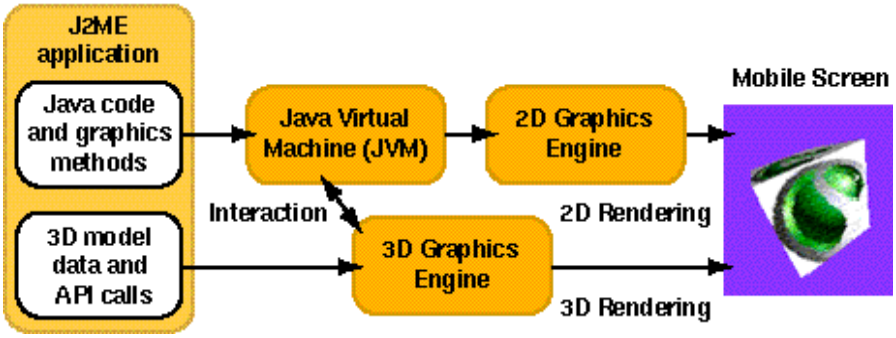
Note: JSR-184 is also enabled in the S302 and W302 series. The W302 series also supports Mascot Capsule Ver. 3.

Java ME integration

In the following text the term "3D engine" refers to either the standard JSR-184 or the Micro3D Ver.3 engine.

The Mascot Capsule 3D graphics engine is integrated seamlessly with the Java ME platform offered in Sony Ericsson phones. To write a 3D MIDlet, first design its control logic and graphics algorithms in Java. Manipulate the 3D engine from a midlet by

directly accessing it via its native Micro3D Ver.3 API or alternatively through the JVM via JSR-184 API. The 3D engine executes in parallel with the mobile phone JVM, as shown in the figure below, to render the 3D image.



No special operations are required to display 3D content with Mascot Capsule Micro3D Ver.3. Simply add calls to its 3D APIs within the Java application.

A developer is not required to do anything special to add 3D capabilities to a Java ME platform MIDlet. Any Java tool preferred by the developer can be used to create MIDlets to craft 3D applications. The 3D engine is accessed via calls to the 3D APIs. No special linking or other post processing is necessary.

Micro3D Ver.3 features

The Micro3D Ver.3 engine has been used in a number of mobile devices around the world. As a consequence, the 3D engine is based on robust, field-tested code. The Micro3D Ver.3 engine performs all graphics operations using 32-bit integers. In short, the engine does not require floating-point hardware or graphics accelerators to function.

A few of the capabilities that the Micro3D Ver.3 engine offers to the Sony Ericsson developer are:

- The ability to access, compose, and display complex 3D objects stored in a resource
- Define surface characteristics for 3D objects such as shading, transparency, and texture maps
- Manipulate objects, such as rotating, scaling, or moving them
- Animation support

These capabilities are discussed later in this document. More information regarding the Mascot Capsule engine can be found at: http://www.mascotcapsule.com/.

JSR-184 – Mobile 3D Graphics API for the Java ME platform

JSR-184 is the standard API for 3D in phones. It can be expected to be included in all phones that support 3D. The engine uses floating point arithmetic in the calculations for highest precision. JSR-184 version 1.1 is supported in Sony Ericsson phones on Java platform JP-8.0 or higher, Sony Ericsson Java platform for Windows Mobile and Sony Ericsson Symbian Java Platform. In these phones JSR-184 has full hardware support.

A few of the capabilities that the JSR-184 offers to the Sony Ericsson developer are:

- The ability to access composite, and display complex 3D objects stored in a resource
- Define surface characteristics for 3D objects such as shading, transparency, and texture maps
- Manipulate objects, such as rotating, scaling, or moving them
- Animation support
- Fog effects
- Morphing effects
- Meshing, including skin meshing
- Texture tiling
- Multiple colored lights.

The article on http://developer.sonyericsson.com/docs/DOC-2484 gives some tips on how to improve performance of applications utilising JSR-184 features.

JSR-239 – Java binding for the OpenGL ES API

Note: The JSR-239 API is supported on the *JP-8.x* platforms and on Sony Ericsson Symbian OS Java platform 5.0 or higher.

JSR-239 defines an optional package, implemented on the CLDC 1.1/MIDP 2 platform.

The OpenGL® ES and EGL APIs are defined by the Khronos Group (www.khronos.org). OpenGL ES defines two profiles: the Common profile and the Common-Lite profile. The Common-Lite profile is a 32-bit fixed-point profile, while the Common profile supports floating point. The Common profile is a superset of the Common-Lite profile. The JSR-239 specification provides bindings to the Common profile (including all fixed-point functions).

All Sony Ericsson JP-8 phones fully support OpenGL and EGL version 1.0. Only RGB565 configurations are currently supported. From JP-8.5 also the OpenGL ES extension pack is supported.

Sony Ericsson Symbian OS Java platform 5.0 or higher, except early released Satio™ phones, support OpenGL ES 1.0.

To define the attributes for an RGB565 configuration:

```
int configAttributes[] = {
     EGL11.EGL_RED_SIZE, 5,
     EGL11.EGL_GREEN_SIZE, 6,
     EGL11.EGL_BLUE_SIZE, 5,
     EGL11.EGL_NONE
};
```

To choose the configuration with these attributes:

```
egl.eglChooseConfig(eglDisplay, configAttributes, eglConfigs,
eglConfigs.length, numConfigs)
```

The complete JSR-239 specification can be downloaded from the Java Community Pages,
http://www.jcp.org/en/jsr/detail?id=239

Sony Ericsson SDK for the Java™ ME platform

The Sony Ericsson SDK for the Java™ ME platform (hereafter referred to as “SDK”), contains:

- Micro3D Ver.3 APIs and API documentation
- JSR-184 APIs, for documentation, see http://www.jcp.org/en/jsr/detail?id=184
- JSR-239 APIs, for documentation, see http://www.jcp.org/en/jsr/detail?id=239
- Micro3D Ver.3 MIDlet emulation
- JSR-184 MIDlet emulation
- device tools - device explorer and device connection proxy tools
- support for On Device Debugging (ODD) for Java ME platform applications. Developers will be able to perform real-time source code debugging of their MIDlets for standard Java ME platform, Micro3D Ver.3, JSR-184, and JSR-239 development.

The Developers guidelines for Sony Ericsson phones provide developers with phone-specific Java technical specifications including phone memory, heap size, canvas size, supported APIs, platform and network capabilities, and so on.

Both the SDK and the Developers guidelines are available at Sony Ericsson Developer World.

Mascot Capsule Micro3D Version 3

This section will describe the Mascot Capsule Micro3D Version 3 in detail. This popular version uses proprietary 3D Java APIs.

Graphics commands

Micro3D Ver.3 graphics API operates in an **immediate** mode, where graphics command are issued into graphics pipeline and the rendering engine executes it. These commands implement the following functions:

- **3D objects descriptions**. These commands describe the coordinates and colours of triangles, polygons, and quad surfaces that comprise the 3D object. They also specify the texture maps and the colour blending processes that are to be applied to each object. Point sprites are also supported
- **3D environment configuration**. These commands adjust the position and characteristics of scene lighting, the scene perspective (parallel or projected), the type of shading to be applied, and whether to use semitransparent blending
- **Manage the rendering process**. These commands describe the type of color blending used in the rendering operation and specify the clipping region coordinates. They also instruct the engine to execute a transformation (such as a rotation or motion) on an object. A flush command indicates when the scene is to be rendered and drawn to the phone screen
- **Graphics primitives**. These commands draw points, lines, triangles, and quads.

Commands can be stored in lists. This allows a 3D object to be stored as a list of commands that specify the object shape, its color, plus its shading and blending characteristics. To draw the object – say, a rocket – the application would issue the command list that stores the rocket description to the Micro3D Ver.3 graphics pipeline. The 3D engine in turn draws the rocket. In Micro3D Ver.3, pixels are drawn to screen only when `Flush()` is called, and not immediately after the command is issued.

Object import

Because of the difficulty to describe any 3D object of moderate complexity using graphics commands, the Micro3D Ver.3 allows import of 3D object data that was generated by 3D authoring programs. This data is stored as a resource in the MIDlet JAR file, and is accessed by the Micro3D Ver.3 `Figure` class. This class contains the

3D object information. Various attributes, such as the object shading type, can be associated with an instance of this class. Other associations specify operations such as rotation, scaling, or motion. Command lists can also be attached to each instance of `Figure`.

Besides simplifying the design of a 3D model, object import also provides two other significant advantages to the developer. First, it reduces the computational overhead of updating a 3D scene. Computations are not required to generate the 3D object itself. Instead, any rendering or transformation operations are applied directly to the polygon information maintained by the instance of `Figure`.

Second, the `Figure` class helps implement animation effects for a model. The current arrangement of 3D objects in a model represents *a pose*. An action description class, `ActionTable`, is associated with the `Figure` and stores motions that change the model pose. The action tables animate the model by changing its pose over time under program control. `ActionTable`s are also imported from a JAR resource.

As a consequence, not only can a 3D authoring program be used to generate 3D objects, the program can also animate them. HI Corporation offers plug-ins for several popular 3D modeling programs that export both the model geometry and animation sequence information into a file format that the Micro3D Ver.3 engine can read. For more information on the plugins and tools, see “Getting started” on page 29.

`Figure`s can be drawn immediately or rendered, that is, all transformations and effects are processed, but the actual generation of pixel data is deferred until a flush command executes. This allows mimicing a 3D retained mode by first arranging and updating the `Figure` objects of a scene, and then performing a flush graphics command. The main reason for "retaining" the rendering commands is that all rendered figures then can be automatically sorted by depth (Z-sorting).

Note that Micro3D Ver.3 does not support the grouping of `Figure`s into leafs and nodes that could assemble a graphics "world", which is a feature of retained mode operation. However, the rendering scheme is suitable for most 3D operations.

Micro3D Ver.3 classes

The Micro3D Ver.3 API consists of ten classes used to manage the display and control of 3D content. Some of these classes store object information – like the `Figure` class introduced above – while others attach attributes to instances of these objects. Other classes describe the lighting and textures used in a scene. One class handles all rendering operations, while a utility class provides methods that can help with the design of graphics algorithms. The table below provides a brief summary of these classes.

Class	Description
`ActionTable`	Stores the action data that controls the movement of an associated 3D model. Action table data can be read from a JAR resource
`AffineTrans`	Handles the matrix math used to scale or move a 3D object

`Effect3D`	Contains the rendering effect data that is associated with a particulat 3D object. Such effects include transparency, shading type and lighting
`Figure`	Stores the geometric information of a 3D object. Also stores pose information. This data is read from a JAR resource
`FigureLayout`	Container for all rendering information of a 3D object, such as its position, size and orientation
`Graphics3D`	Implements all rendering functions. Must be bound to a LCDUI Graphics object in which to draw the pixels
`Light`	Contains lighting information, such as its direction and intensity
`Texture`	Stores the texture data for objects or the environments (background). The data can be read from a .bmp file stored as a JAR resource
`Util3D`	Contains utility methods for use in 3D algorithms, notably `sin()`, `cos()` and `sqrt()`
`Vector3D`	Contains methods that construct 3D vectors or extracts information (such as the vector x, y, and z components) from them. Also implements some vector math routines

Detailed information on these classes and their methods can be obtained from the JavaDoc reference that accompanies the Sony Ericsson SDK for the Java™ ME platform.

Mobile 3D Graphics API for the Java ME platform (JSR 184)

This section describes in detail the Mobile 3D Graphics API for the Java ME platform (called JSR-184). For more information about this standardised API, see http://www.jcp.org/en/jsr/detail?id=184.

Graphics commands

JSR-184 supports **retained** and **immediate** modes of scene rendering. The advantage of JSR-184 retained mode is that it allows less 3D-savvy developers to readily build complex 3D models. The retained mode simplifies the design of a 3D world by hiding lots of low-level gritty details.

Unlike Micro3D Ver.3, JSR-184 does not implement graphics commands that can be stored into lists and poured into the graphics pipeline.

Because the retained mode capabilities are built on top of the immediate mode operations, the retained mode also benefits from hardware acceleration. This design approach also allows the simultaneous use of both retained and immediate mode operations with JSR-184. The ability to use both modes allows a programmer to put the limited resources of the phone to their best possible use, rendering speed versus resource consumption can be effectively balanced by using the appropriate mode for different 3D graphics tasks.

Note that JSR-184 was designed so that it could be implemented on top of OpenGL ES. The Sony Ericsson JSR-184 implementation does not support OpenGL ES commands.

Object import

JSR-184 allows importing 3D content that contains geometric object data and animation information. Object data can also carry attribute information, such as material, fog, texture, compositing mode, and others. This 3D content can be generated on PCs and workstations using 3D authoring programs and, using a plugin translator, export the content into a .m3g file. The JSR-184 specification contains a detailed description of this file format. The file can be stored as a JAR resource in the

application, where it is read by the `Loader` class. The `Loader` generates instances of 3D objects as it reads the resource. Entire scene graphs with animation, lighting, and camera views can be imported and displayed.

Because the format of the .m3g file is an open standard, it enables 3D content to be generated on high-end platforms, then readily imported or shared among a wide variety of mobile platforms that implement JSR-184.

JSR-184 classes

Compared to Micro3D Ver.3, JSR-184 contains lots of classes, thirty in all, as in the table below. The large number of classes increase the implementation firmware footprint. The APIs have been designed to reduce RAM use by storing most objects by reference, not by making new instances of them.

JSR-184 supports both **retained** and **immediate** modes of operation. The retained mode uses scene graph that links all of the geometric objects in the 3D world via a structured tree of nodes. Each node on the graph represents a geometric object and carries information about its appearance, position in space, and how it behaves in relation to other nodes. Objects subclassed from `Node` are used (such as `Light`, `Sprite`, and `Mesh` objects) to assemble the 3D world. The `Group` class allows gathering of unordered `node` objects together, and the `World` class defines a special `Group` node that acts as the top-level container for all of the nodes in the 3D world. To display a view of the 3D world using the retained mode, a Graphics3D `render` method is executed on the `World` node.

Two classes, `TriangleStripArray` and `VertexArray`, serve as building blocks from which more complex 3D objects and models can be assembled. Other subclasses of `Node`, such as a `Mesh`, are also used to build more complex 3D geometric objects. Other `Node` subclasses control the scene graph lighting and point of view (`Light` and `Camera`, respectively). There are classes that define the visual properties of a 3D object (`Material`, `Fog`, `CompositingMode`, and others) and they can be used in either the immediate mode or retained mode.

A detailed description of these classes is out of the scope of this document. For more information, consult the JSR-184 specification documentation at http://www.jcp.org/en/jsr/detail?id=184. The table below provides a brief summary.

Class	Description
`AnimationController`	Manages the location and speed of a collection of objects that comprise an animation sequence.
`AnimationTrack`	Contains the information that controls a single animation property on one target object. Animation sequences consist of a set of `AnimationTracks` handled by an `AnimationController`.
`Appearance`	Stores the rendering attributes of a set of component objects. The attributes describe material characteristics of each object, its polygons, how it is to be blended into the scene, and any fog effects. It also specifies texture map characteristics and the images involved.

`Background`	Used to specify a color or image that clears or fills the given `viewport` (a drawing area).
`Camera`	The node in a scene graph that establishes the point of view of the scene. Used to render the scene graph from 3D to 2D. It also establishes what elements of the scene graph are visible (clipping).
`CompositingMode`	An `Appearance` component that contains the attributes used in pixel compositing operations.
`Fog`	An `Apperance` component that contains the attributes used to apply a fog effect to pixels.
`Graphics3D`	A graphics context that is applied when rendering an image. This object binds to a rendering target, which is `Canvas`, mutable `Image`, or a `CustomItem`. The rendering target receives the rendering operation output. This class handles all of the drawing for this API.
`Group`	A scene graph node that stores a collection of unordered nodes as its children.
`Image2D`	Stores a two-dimensional image that can be used as a texture, a background, or a sprite image.
`IndexBuffer`	Describes how to connect vertices so that they assemble a geometric object. As an abstract class, `TriangleStripArrays` is used to construct objects.
`KeyFrameSequence`	Stores animation data. The data is stored as a sequence of time-stamped, vector-valued keyframes. Each keyframe stores the value of an animation characteristic at a given moment in time.
`Light`	A scene graph node that represents a light source. Lights have color, intensity, and a type (`ambient`, `directional`, `omnidirectional`, and `spot`).
`Loader`	Downloads node components, scene graph nodes, and entire scene graphs for use in graphics operations. The data can contain `Camera` and `Light`, `Appearance` and `Material` attribute classes, and animation classes such as `AnimationTrack` and `KeyFrameSequence`. Used to import ready-made 3D content.
`Material`	An `Appearance` component that stores material attributes used in lighting calculations.
`Mesh`	A scene graph node that represents a 3D object. The object is described with polygonal surfaces. A `Mesh` consists of triangle strips defined in an `IndexBuffer` instance, along with its visual properties, which are stored in instances of `Appearance`.
`MorphingMesh`	A scene graph node that represents a vertex morphing polygon mesh. `MorphingMesh` is similar to `Mesh`, but its vertices are calculated using weighted values in `VertexBuffers`. This enables a `MorphingMesh` object to change its shape.
`Node`	An abstract base class for all scene graph nodes. `Camera`, `Group`, `Light`, `Mesh`, and `Sprite3D` are all sub-classed from it.
`Object3D`	An abstract base class for all objects that can be part of a 3D world.

`PolygonMode`	An `Appearance` component that contains polygon-level attributes. These attributes include back/front face culling, lighting computations, perspective corrections, shading, and winding.
`RayIntersection`	An object that contains rays added to it by the `Group` `pick()` method. `RayIntersection` stores references to the meshes or sprites that intersect each ray, and information about the intersection point.
`SkinnedMesh`	A scene graph node that represents a skeletally animated polygon mesh. Allows groups of vertices to change independently of one another while smoothly deforming the polygon mesh. Provides an efficient means of animating characters.
`Sprite3D`	A scene graph node that represents a 2D image with a 3D position.
`Texture2D`	An `Appearance` component that stores a 2D texture image and the attributes that direct how the image is applied to submeshes.
`Transform`	A 4-by-4 matrix that contains values used to perform a transformation operation.
`Transformable`	An abstract base class for `Node` and `Texture2D`. It defines common methods used to handle node and texture transformations.
`TriangleStripArray`	Defines an array of triangle strips that are used to construct a geometric object.
`VertexArray`	An array of integer vectors that represent vertex positions, normals, colors, or texture coordinates.
`VertexBuffer`	Contains references to the `VertexArrays` that define the characteristics for a set of vertices.
`World`	A special `Group` node that is the top-level container for scene graphs.

Detailed information on these classes and their methods can be obtained from the JavaDoc reference that accompanies the Sony Ericsson SDK for the Java™ ME platform.

Getting started

The Java ME platform execution environment integrates both the JSR-184 and Micro3D Ver.3. This implies that familiar Java development tools are used to write a 3D application. However, the generation of the files that Mascot Capsule imports to assemble and render 3D models is a more involved process, described below.

Micro3D Version 3 model conversion

There are numerous commercial 3D authoring tools on the market that are used to generate 3D models and animation for PC application or game consoles. Normally, the model and animation data generated by these tools are not compatible with the JSR-184 or Micro3D Ver.3 environment. Examples of these 3D modeling development tools are ***3ds max***, ***LightWave*** and ***Maya***.

HI Corporation offers a number of plugins that can be used from within these authoring tools to export the model and animation data to intermediate files. These files can be subsequently converted to Micro3D object and action format files. This allows an application designer who created 3D models for game consoles to leverage the same authoring tools and expertise to make 3D content for mobile applications.

After the model and action data are exported into their intermediate formats, several utility tools for post processing and final translation can be used. The ***PAC.exe*** utility allows viewing and adjusting the attributes of certain types of polygonal data in the intermediate model data file. A converter utility application, ***Micro3D_Converter.exe,*** translates the intermediate 3D data and action files into file formats that Micro3D can use. A ***PVMicro.exe*** application allows viewing of the 3D models in their native Micro3D Ver.3 3D format for final check out.

The table below summarises the file types involved in the export/translation procedure and the file types involved in the export and translation of 3D model and animation data for consumption by Micro3D Ver.3

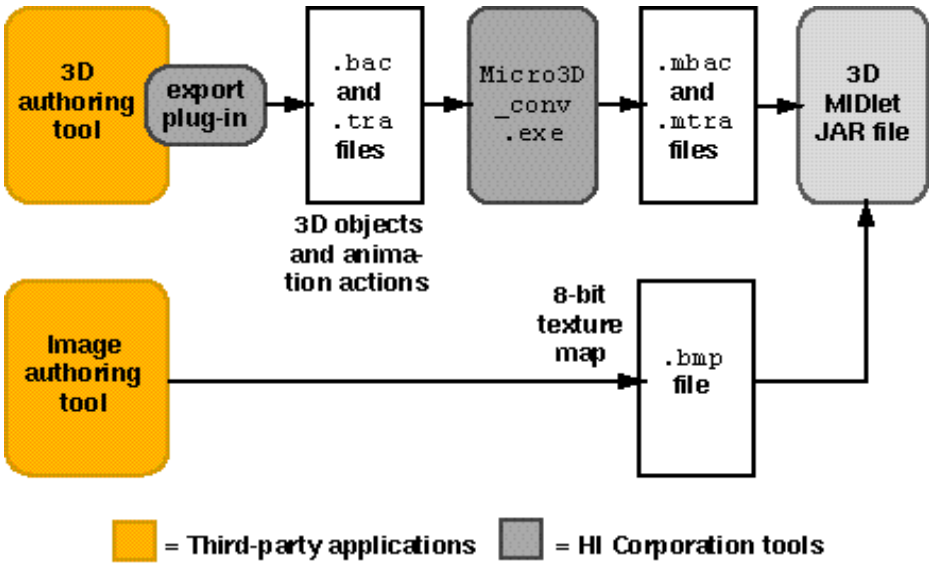
File extension	Description
.bac	Intermediate file that contains the 3D model data exported from the 3D authoring tool
.tra	Intermediate file that contains animation (action table) data exported from the 3D authoring tool
.mbac	Final file that contains the 3D model data formatted for use in Micro3D
.mtra	Final file that contains the 3D action table data formatted for use in Micro3D
.bmp	Stores 8-bit texture image data

All of the plugins, tools, and the documentation necessary to manage the export/ translation process are available from Mascot Capsule Web site at http://www.mascotcapsule.com/toolkit/sony_ericsson/. These tools are provided by HI Corporation and all feedback and trouble reports shall be submitted to this website.

Note: It is important that the version number of the 3D modeling program used and the export plugin version number match.

Conversion flow for Micro3D Ver.3

1. Create the 3D model using the various commercially available 3D authoring tools.
2. Export the data from the 3D authoring application using the available plug-ins. The plug-in outputs intermediate 3D format files (.bac and .tra).
3. Optional: Change and edit polygon attributes in the .bac file using the ***PAC.exe*** tool.
4. Use the ***Micro3D_Converter.exe*** tool to translate the intermediate format files (.bac and .tra) into Micro3D Ver.3 format files(.mbac and .mtra).
5. Optional: Preview the 3D models in the final files(.mbac and .mtra) with the ***PVMicro.exe*** tool.
6. Add the final files(.mbac and .mtra) as resources to the 3D MIDlet JAR file.
7. Add texture maps (.bmp files) to the 3D MIDlet JAR file. The images must use 8-bit pixels (256 colors).



Java ME platform application set up for Micro3D Ver.3

This section describes how to properly initialise the Micro3D graphics environment within a mobile application and display 3D graphics with it. Briefly, the proper sequence of events to configure and use the Micro3D Ver.3 graphics engine is as follows:

1. Load the 3D model data.

2. Initialise any attributes for the 3D objects of the model.

3. Set up the camera.

4. Set up the light and any effects.

5. Initialise the graphics environment, render the model, and display it.

Load the 3D model data

As stated above, it is far easier to assemble the 3D model by importing 3D object data. More precisely, when an instance of a `Figure` is made, its 3D geometric information and pose is imported from a `.mbac` file. Any required animation data must be imported from a `.mtra` file and stored in an instance of an `ActionTable`. Finally, if the model uses texture maps, they are imported from a `.bmp` file. All of these files are stored as JAR resources in the MIDlet.

For each 3D object, the following code applies:

```
// Declare the 3D model elements

Texture texture;
Figure figure;
ActionTable action;

// Load (import) the 3D object data from JAR resources

try {
   texture = new Texture( "/texture.bmp", true);
   figure = new Figure ( "/figure.mbac" );
   action - new ActionTable( "/action.mtra" );
} catch (Exception e) {}
```

As the code sample shows, importing the data makes the whole process of constructing the 3D model straightforward and easy.

Initialise the 3D object attributes

The figure geometry, some action information for it, and its textures are loaded. To associate the action information and texture map with the figure:

```
//Use this texture with the 3D object
figure.setTexture( texture );
```

```
// Associate the action table with this object
figure.setPosture ( action, 0, 1 );
```

The arguments to the `setPosture()` method associate the `action` table action with `figure`. The animation starts with the first frame of the model and the first action command from the table.

Set up the camera

By positioning the viewing camera, the point of view of the scene has been set up. To set up the camera viewing angle vectors:

```
Vector3D position = new Vector3D( 0, 100, 256 );
Vector3D look = new Vector3D( 0, -2048, -4096 );
Vector3D up = new Vector3D( 0, 4096, 0 );
AffineTrans trans = new AffineTrans();
trans.lookAt( position, look, up );
```

These operations create vectors, then feed them to a special `AffineTrans` method, `lookat()`. This method applies the supplied vectors to matrix transformations and generates the scene geometry from the specified point of view.

Besides specifying the viewing angle, other attributes need to be associated with the 3D object. The `FigureLayout` class acts as a container for certain figure attributes, such as the `Figure` point of view (POV) transform, plus scaling, position data, and perspective type. The `FigureLayout` instance is initialised as follows:

```
FigureLayout layout;  // Make an instance

// Set the distance between the 3D model and the camera.
layout.setPerspective( 1, 4096, 512 );

// Set where the camera's center is positioned in the display.
layout.setCenter( 128, 128 );

// Set the camera angle using the supplied parameters.
layout.setAffineTrans( trans );
```

With the layout set up, the scene lighting is adjusted.

Set up light and any effects

A Micro3D Ver.3 scene has two lights: a spotlight and an ambient light. Both are white, so no color descriptions are possible. The following code sets up a direction vector for the spotlight, and configures the intensity of the two lights:

```
Vector3D lightDirVec = new Vector3D( -146, 293 439 );
int lightDir = 3730;
int lightAmbient = 1626;

// Make instance of Light from the prepared parameters
Light light = new Light( lightDirVec, lightDir, lightAmbient );
```

These light characteristics are added to an instance of `Effect3D`:

```
Effect3D effect;
effect.setLight( light );
```

Note that other attributes could be added to the instance of effect, such as shading types and whether color transparency is enabled or not.

Initialise the graphics environment, render the model, and display it

With the `Figure` and many of its attributes prepared, it should be rendered and displayed. The `Graphics3D` class is used for this, since it contains all of the rendering methods. However, first the output of this class must be "bound" to a LCDUI `Graphics` object so that the final results appear on screen. Then the scene is drawn, and finally the `Graphics3D` instance is released. The following code shows how this is done:

```
Graphics3D g3d = new Graphics3D();
Graphics    g;
// Bind the 3D graphics context to the MIDP Graphics object.
    g3d.bind(g);

try {
    g3d.renderFigure( figure, 80, 100, layout, effect );
    g3d.flush(); // Draw the rendered figure on-screen
} catch( Exception e ) {}

// Release the MIDP Graphics object.
    g3d.release( g );
```

Note how the `renderFigure()` method allows a model (`figure`) to be supplied along with its layout information (`layout`), and any special effects (`effect`). These attributes are applied to the `Figure` during the rendering process. The `flush()` method pushes the generated pixels onto the screen.

Mobile 3D Graphics API (JSR-184)

The same 3D modelling tools used for Micro3D Ver.3 can be used to create data used in Mobile 3D Graphics Version 4. (m3g)

HI Corporation offers a number of plug-ins that can be used from within these authoring tools to export the model and animation data to intermediate files.

The table below summarises the file types involved in the export/translation procedure and the file types involved in the export and translation of 3D model and animation data for consumption by Micro3D engine Version 4 (JSR-184)

File format	Description
H3T	Intermediate 3D data exported from 3D authoring tool
M3G	The 3D data format used in JSR-184
PNG	The file format for texture

After the model and action data are exported into their intermediate formats, several utility tools for post processing and final translation can be used. The ***V4Converter.exe*** utility allows converting of H3T data to M3G data. A ***V4Examiner.exe*** tool allows viewing and verifying of M3G/H3T files.

All of the plugins, tools, and the documentation necessary to manage the export/ translation process are available from the Mascot Capsule Web site at http://www.mascotcapsule.com/M3G/download/e_index.html. These tools are provided by HI Corporation and all feedback and trouble reports shall be submitted to this website.

References

Useful links

- Sony Ericsson Developer World (http://www.sonyericsson.com/developer)
- Mascot Capsule Toolkit for Sony Ericsson phones (http://www.mascotcapsule.com/toolkit/sony_ericsson)
- Mascot Capsule Micro3D V4 - m3g Tools and Plug-ins (http://www.mascotcapsule.com/M3G/download/e_index.html)
- Mobile 3D Graphics API for the Java ME platform (JSR-184) (http://www.jcp.org/en/jsr/detail?id=184)
- Java binding for the OpenGL ES API (JSR-239) http://www.jcp.org/en/jsr/detail?id=239
- Discreet 3ds max (http://www4.discreet.com/3dsmax/)
- Newtek Lightwave 7.5 (http://www.newtek.com/products/lightwave/product/7.5/)

Sony Ericsson 3D Knowledge Base

Sony Ericsson phones offer hardware based 3D support through Mascot Capsule technology. Mascot Capsule delivers 3D support in two flavours, version 4 and version 3, which provides JSR-184 and a high performance proprietary API respectively.

This knowledge base attempts to provide an extensive list of answers to common questions encountered when developing 3D content that uses Mascot Capsule technology.

General

Q: The Sony Ericsson phones use Mascot Capsule as a 3D engine, how accurately is this reflected in the SDK?

A: The emulator uses only a slightly modified version of Mascot Capsule designed for use on a desktop environment. In essence the accuracy is very high.

Q: Is correct timing for 3D applications implemented in the emulator?

A: To simulate the speed of the actual phone, the "Graphics Latency" preference can be tuned in the Sony Ericsson WTK2 Preferences under the Performance tab. However, beware that input will lag behind as well.

Q: My exported 3D Max 7 models are very large, how can I reduce size?

A: Textures usually consume the most space and a good starting point is to try to reduce the number of textures being used. Another consideration is that the H3T exporter that HiCorp provides for Maya, 3DStudio and LightWave always exports TrueColour PNGs. Ensure that PNG files are optimised after a H3T file is exported.

Q: When attempting to run a MIDlet that uses 3D functionality I receive an exception with "Couldn't load zayitlib.dll" as a root cause?

A: This represents the Mascot Capsule 3D engine used within the SDK to accurately reflect the phone implementation. This error indicates that the library could not be correctly registered during the installation process. Close all applications and attempt to reinstall the SDK again to rectify the problem.

Q: If I load multiple M3G files sequentially, after a time I get a "throw dummy IOException" even though all references are being nulled and System.gc is being called?

A: This is a known issue in early Z500 firmware and has been fixed in later revisions.

Q: Which Sony Ericsson phones support Mascot Capsule Micro3D and JSR-184?

A: See “Products” on page 3 for a list of phones.

Q: Where can I get Mascot Capsule Version 3 Javadoc?

A: The API Javadoc is included in the Sony Ericsson SDK for the Java™ ME platform.

Q: Where can I get JSR-184 Javadoc?

A: The JSR-184 Javadoc is included in the Sony Ericsson SDK for the Java™ ME platform and is also available at http://jcp.org/aboutJava/communityprocess/review/jsr184/index.html

Q: What support for Java 3D development is provided by the Sony Ericsson SDK for the Java™ ME platform?

A: The SDK includes:

- Mascot Capsule Version 3 API and Javadoc
- Mascot Capsule Version 4 API (Mobile 3D Graphics API for Java ME, JSR-184) and Javadoc
- On Device Debugging for Mascot Capsule Version 3 and 4 (JSR-184).

Mascot Capsule Version 3

Q: How do I build my project in the SDK ?

A: Create a new project in Ktoolbar. Select *Project->Settings* menu, on *API Selection* tab choose "Custom" platform. Check the appropriate checkbox "Mascot Capsule Ver.3 API". Now you will be able to build your project with support of Mascot Capsule V3 API.

Q: What tools are available to create models?

A: All Micro3D conversion tools are available at Mascot Capsule website http://www.mascotcapsule.com/toolkit/sony_ericsson.

Q: What texture sizes can be used ?

A: Unlike Mascot Capsule Version 4, arbitrary sizes can be used for example, 64 x 192. The texture image file needs to be less than 256x256 pixels and use 8 bits (256 colours).

Q: I cannot position a model on the phone using the Z axis ?

A: Be sure to define the Z-axis position. See the method `setPerspective()` in the `FigureLayout` class. Perhaps parallel projection is being set instead of perspective projection?

Q: There appears to be no support on for defining and rendering your own geometric model to the screen.

A: Refer to `Graphics3D::renderPrimitives`.

Q: Can I render a scene to an off-screen object ?

A: Yes. Use the `bind()` method in `Graphics3D()` to direct the 3D drawing to the graphics object `javax.microedition.lcdui.Image::getGraphics()`.

Q: Which graphics format is supported by Mascot Capsule v3?

A: Mascot Capsule v3 only supports uncompressed 256 colours indexed BMP. BMP RLE compression is not supported for creating textures.

Q: When using the `AffineTrans setRotation` method, what is does the integer parameter indicate ?

A: The rotation angle is specified as an integer assuming 360 degrees = 4096. The rotation is a clockwise rotation looking in the positive axis direction.

Q: What are the .bac and .mbac files and what is their relation?

A: The .bac file is generated by the 3D modeling tooling using the Mascot Capule version 3 plugin. This file represents your model in the propriety Mascot Capsule format. The .bac file is intended to be used with the Micro3D Converter tool provided by Mascot Capsule which in turn generates a .mbac file that is optimised for a small foot print and usage on a phone.

Note that the binary (bac) file format cannot be used to gain access to triangles (for advanced collision detection and direct triangle transformation). For this, use the ascii format. It is important to know which models can be represented by ascii files and which cannot.

Q: How can I do ToonShading

A: ToonShading, or CelShading, cannot be performed by the hardware, since the current hardware on all 3D-enabled phones does not support direct manipulation of the graphics pipeline (shading). However, one can mathematically create CelShaded models in software. It is done by manually plotting pixels to the graphic buffer after all models have been rendered. This technique is very slow unless done right. The key is to work with unshaded models with no (or almost no) textures and later shade them directly onto the graphics buffer with pixel-plotting.

Q: Is there any support for perspective correction ?

A: The FigureLayout class provides a convenient setPerspective method which can be used for perspective correction.

Q: Is it possible to export several 3D objects in a scenario at to one Bac file?

A: Yes, but it is recommended to export them as individual Bac files. This allows your objects to be individually loaded in your application. Also, this minimises errors with loading files onto phones. Some phones might throw exceptions when too complex files are loaded.

Q: What do the constants `PATTR_BLEND_NORMAL`, `PATTR_BLEND_ADD`, `PATTR_BLEND_HALF` and `PATTR_BLEND_SUB` mean?

A: The commands are used to blend polygons. The table below shows how the polygon background is mixed with the texture.

The RGB colors from the background are mixed with the RGB colors from the texture individually. Half, Addition and Subtract cannot be used at the same time for the same polygon.

Attribute	Background		Texture
Normal	0%	+	100%
Add	100%	+	100%
Half	50%	+	50%
Sub	100%	-	100%

Q: Is there any way to access the vertices of a 3Dobject in my MIDlet using Mascot Capsule version 3? If not, how should I check whether the model touches the ground?

A: To check how a model touches the ground is really a pure polygon-collision issue. Most of the time a developer will create some kind of level editor, where clear boundaries are made for all the walkable objects (usually large bounding boxes). Thus one doesn't have to have vertex access but can instead mathematically determine if one object is touching another. All that is needed is a file containing all ground bounding boxes and a bounding box or sphere for your model.

To create a bounding sphere for a model only two single variables need to be known. The center of the model, and the radius of the sphere. Most developers export bounding boxes when the model is converted, by reading an ascii format. That way one has no need for direct vertex manipulation either.

Q: How can I make a racing game MIDlet using Mascot Capsule version 3, when the maximum value for the far-clipping plane is 32767? For accurate sine and cosine as well as for vector analysis I normally have to scale the scene so that 4096 represents the 1.0. If I do so, I only have 7 units for my viewing frustum, difficult for a racing game.

A: While there are many tricks you can use to make a racing game in fixed-point range, the first step is to start sacrificing accuracy, most of which would be lost during the rasterisation step (since the viewpoint dimensions are less than 200 pixels).

Q: Is there any possibility to emulate fog ?

A: Yes, however, fog involves an extra color blending operation per pixel, making it very slow to implement in software. There are many ways to emulate fog, using semi-transparent textured triangles is one of them.

Q: Which opacity (transparency) settings can I use for texture on 3D objects ?

A: There are 3 opacity settings that can be used: 0%, 50% and 100%. If one wanted to alter the colour of a texture by simply changing the material colour of the model, one can do so by choosing the 50% setting.

Q: Is there any methods to applicable to 3D collision detection ?

A: Unlike Mascot Capsule v4, where there's an implemented method for Ray Intersection, version 3 does not support this.

Q: In order to implement a correct collision detection framework we need a function to retrieve all vertices for a Figure in Mascot Capsule Version 3. How can we achieve that?

A: If you choose to use `renderPrimitives()`, you have the geometry data. When using `Figures` loaded from a resource there is no way to retrieve the geometry. However, in practice you would have some more suitable representation of the objects for collision detection, typically a hierarchy of spheres, OBBs, and perhaps – if really necessary – a simplified version of the rendered geometry. Alternatively, you can keep track of the vertices in your game and draw the scene using the 3D graphic.

Note that instead of using the binary format (bac), by using the ASCII format, vertex data and model data can be stored.

Q: Is it possible to use tiling (looping) textures on 3D models in a MIDlet?

A: By tiling textures in your favorite 3D tool and exporting them with HiCorp's tools, you will get the desired tiling effect. However, this isn't always effective and should be used with caution. For generating floors and ground, use the renderPrimitives method with textured triangles or quads.

Q: Is there any limit on the number of polygons that can be used within a scene ?

A: An unlimited number may be used. However, performance will be a limiting factor.

Q: There seems to be no cull functionality in micro 3D? Is it possible to specify back face culling?

A: It is possible to do back face culling and it can be set directly to the figure data by using the PAC tool.

The PAC tool can be downloaded from
http://www.mascotcapsule.com/toolkit/sony_ericsson

Q: Why doesn't the model display in the viewer?

A: The UV values of bac data are all whole numbers. If decimals appear in the UV coordinate values of the 3D tool, deviations will occur when exporting to the bac file. The extent of the error will only be plus or minus one pixel, but this is sufficient enough to distort textures. Currently, there is no solution to this problem. However, as one method of improving the appearance of your work, we recommend that eyes and alphanumeric characters, for example, use higher resolution textures. In addition, fine textures may run together when displayed on an actual phone, if sizes are too small. This is caused by lower screen resolutions on phones.

Another consideration is exporting models without any camera or lights which might cause the viewer to display them incorrectly.

Mascot Capsule Version 4 (JSR-184)

Q: Which tools and documents are available to create Mascot Capsule Version 4 (JSR-184) models?

A: All offical Mascot Capsule created tools and plugins can be found at http://www.mascotcapsule.com/M3G/download/e_index.html.

A number of projects are underway by various parties, some popular links are:

- http://sourceforge.net/projects/juinness/
  Finnish project dedicated to creating a M3G converter that will be able to read almost any common 3D format
- http://fivedots.coe.psu.ac.th/~ad/jg/objm3g/
  A tutorial explaining how to convert OBJ files to M3G
- http://www.renderware.com/
  Renderware, a very well-known program throughout the industry
- http://www.jcodeworks.com/demomidp.html
  A M3G exporter that takes OBJ files and turns into a format that can be read by the supplied loading class.

Q: When loading a M3G a "nested uri" IO exception is thrown?

A: This occurs when using multiple maps and textures that use more than 256 colours. Ensure your texture maps use a maximum 256 colours palette. Under 3D Max, there is a 'Texture Tool' within the exporter dialog with an option to use 'Optimized Palette' which will facilitate this.

Q: When exporting using the native 3D Max M3G plugin I note that the output is larger than when using the Mascot Capsule exporter plugin?

A: The 3D Max M3G plugin does not perform any optimisation in regard to calculating triangle stripes in contrast to the Mascot Capsule exporter plugin and therefore produces potentially larger file sizes. We recommend using the HiCorp exporter plugin.

Q: Are there any methods applicable to 3D collision detection?

A: No, There is no API defined for collision detection. However, the `RayIntersection` class computes a ray intersection in the 3D world.

Q: What texture sizes can be used?

A: All textures in JSR-184 must be of the size 2^n x 2^n (for example 32x32, 128x64). Recommended maximum size is 256 x 256 (256 x 256 texture is usually the uppermost limit when it comes to heap memory).

Q: When I scale my model in 3D Max using the scale tool and then export it for preview, it does not render as expected.

A: The 3D Max scale tool does not operate on the object vertices and instead applies scaling to the transform matrix. You should scale the vertices directly in order to ensure that the preview in 3D Max closely reflects the Mascot Capsule render. It is also possible to scale objects on the fly through the Mascot Capsule v4 API.

Q: I export my model from 3D Max and it does not render as expected with Mascot Capsule?

A: Remember to keep the modifier stack short and collapse it before exporting. Some modifiers may hold materials inside which will not be visible in 3D Max but will appear when rendered by Mascot Capsule. The M3G Viewer can also be invaluable during the exporting process. Test all files in the viewer before converting and inserting into the game.

Q: Is it possible to use tiling (looping) textures on 3D models?

A: Yes, tiling textures are supported.

Q: Is there any way to change the vertices directly for an object instead of having to change to position array and set that to the VertexArray. It would be better to have a reference to the positions directly. I think the set method is copying the primitive array.

A: Unfortunately the only way to reference the vertices of an object, is by copying the ones existing in the `VertexArray` of the object, create a new array, delete the old one and perform a garbage collection. This is not something that one wants to do each frame. However, sometimes it is a must, because models tend to deteriorate after many transforms have been applied to them. This is when one must re-load the model into memory, to retain its original vertex coordinates.

Q: When I apply my second texture to my figure, the figure disappears? How can I use more than one texture?

A: You have to specify texture coordinates for each texture to be used on the object:

```
vertexBuffer.setTexCoords(0, TEXTURE_ARRAY, (1.0f/255.0f),
null);
```

```
vertexBuffer.setTexCoords(1, TEXTURE_ARRAY, (1.0f/255.0f),
null);

appearance.setTexture(0, texture); // add the texture to the
appearance.
apperance.setTexture(1, texture2); // add the second texture to
the appearance
```

Q: I have specified three different colors for a triangle (one for each vertex) but it only shows one color. Why?

A: To show all three colors and get a smooth shading you have to specify the `SHADE_SMOOTH` parameter for the `PolygonMode` that you are using.

For example:

```
public Mesh createTriangle() {
        short []POINTS = new short[] {-1,-1, 0,
                                       1,-1, 0,
                                       0, 1, 0};

        int []INDICES = new int[] {0, 1, 2};
        byte []COLORS = new byte[] {127, 0, 0,
                                    0, 127, 0,
                                    0, 0, 127};


        int []LENGTH = new int[] {3};

        VertexArray POSITION_ARRAY, COLOR_ARRAY;
        IndexBuffer INDEX_BUFFER;

        POSITION_ARRAY = new VertexArray(POINTS.length / 3, 3,
2);
        POSITION_ARRAY.set(0, POINTS.length / 3, POINTS);
        COLOR_ARRAY = new VertexArray(COLORS.length / 3, 3, 1);
        COLOR_ARRAY.set(0, COLORS.length / 3, COLORS);
        INDEX_BUFFER = new TriangleStripArray(INDICES, LENGTH);

        VertexBuffer vertexBuffer = new VertexBuffer();
        vertexBuffer.setPositions(POSITION_ARRAY, 1.0f, null);
        vertexBuffer.setColors(COLOR_ARRAY);
        Mesh mesh = new Mesh(vertexBuffer, INDEX_BUFFER, null);

        Appearance appearance = new Appearance();
        PolygonMode polygonMode = new PolygonMode();
        polygonMode.setPerspectiveCorrectionEnable(true);
        polygonMode.setCulling(PolygonMode.CULL_NONE);
        polygonMode.setShading(PolygonMode.SHADE_SMOOTH);
        appearance.setPolygonMode(polygonMode);
        mesh.setAppearance(0, appearance);
        return mesh;
    }
```

Q: I have created a triangle, but when I rotate the plane it disappears after rotating 90 degrees and remains invisible until it has rotated 270 degrees. The backside of the triangle is black ?

A: This is a method called back culling and it is used to gain performance by showing only the front side of each polygon. You can specify the culling in the `PolygonMode` class:

```
PolygonMode polygonMode = new PolygonMode();
polygonMode.setCulling(PolygonMode.C ULL_BACK);

appearance.setPolygonMode(polygonMode);
```

Q: How can I set texture coordinates on a figure with multiple vertices when the `VertexArray` doesn't allow floating numbers and the texture coordinates must have values between 0 and 1?

A: Enter your texture coordinates with values from 0 to 255. When you apply your texture coordinates to the `VertexBuffer`, you scale them down by 1.0f/255.0f:

```
vertexBuffer.setTexCoords(0, TEXTURE_ARRAY, (1.0f/255.0f),
null);
```

Do not forget that the texture maps width and height must be a non-negative power of two (2, 4, 8, 16, 32, 64, 128, 256).

Q: Should the objects in the .m3g files be compressed or uncompressed?

A: When .m3g files are shipped inside the application .jar file, it is preferrable to leave the object data uncompressed. This is because .jar files are compressed anyway so compressing the object data would just give the phone the overhead of loading the compressed m3g while gaining nothing in terms of file size.

Q: Is it possible to just specify eight vertices for a cube and still specify the right texture and normal coordinates?

A: It is possible to draw a cube where three faces share the same vertex and set colors to it. But texture and normal coordinates requires unique values for each vertex, which means that you must specify four vertices for each face on the cube.

Specify eight points for the cube and then use an `IndexBuffer` to specify how to draw the triangles, remember that each face of the cube is built of two triangles.

For example:

```
short POINTS[] = new short[] {1, 1, 1,  // 0
                              1,-1, 1,  // 1
                             -1, 1, 1,  // 2
                             -1,-1, 1,  // 3
                             -1, 1,-1,  // 4
                             -1,-1,-1,  // 5
                              1, 1,-1,  // 6
                              1,-1,-1}; // 7
```

```
int INDICES[] = {2, 3, 0, 1, // Front Face
                 4, 5, 2, 3, // Left Face
                 6, 7, 4, 5, // Back Face
                 0, 1, 6, 7, // Right Face
                 4, 2, 6, 0, // Top Face
                 3, 5, 1, 7};// Bottom Face

int[] LENGTHS = new int[] {4, 4, 4, 4, 4, 4};

POSITIONS_ARRAY = new VertexArray(8, 3, 2);
POSITIONS_ARRAY.set(0, 8, POINTS);
INDEX_BUFFER = new TriangleStripArray(INDICES, LENGTHS);
```

To specify the texture and normal coordinates we must use four points for each face of the cube:

```
short []POINTS = new short[] {-1, -1, 1,      1, -1, 1,     1, 1,
1,      -1, 1, 1, // Front
                              1, -1,-1,     -1, -1,-1,    -1, 1,-
1,       1, 1,-1, // Back
                             -1, -1,-1,     -1, -1, 1,    -1, 1,
1,      -1, 1,-1, // Left
                              1, -1, 1,      1, -1,-1,     1, 1,-
1,       1, 1, 1, // Right
                             -1,  1, 1,      1,  1, 1,     1, 1,-
1,      -1, 1,-1, // Top
                             -1, -1,-1,      1, -1,-1,     1,-1,
1,      -1,-1, 1};// Bottom

short []TEXTCOORDINATES = new short[] {0, 1,      1, 1,       1,
0,       0, 0,
                                       0, 1,      1, 1,       1,
0,       0, 0,
                                       0, 1,      1, 1,       1,
0,       0, 0,
                                       0, 1,      1, 1,       1,
0,       0, 0,
                                       0, 1,      1, 1,       1,
0,       0, 0,
                                       0, 1,      1, 1,       1,
0,       0, 0};
```

Q: Camera alignment does not update for a given Node.

A: The node must be rendered for camera alignment to take place

Q: Why are some polygons in my scene flickering?

A: Micro3D uses the Z-sort method for each individual polygon. As a result, overlapping polygons may flicker when rendered. We recommend models that avoid overlapping as much as possible.

Q: Is my understanding correct in that to render polygons on the phone, I must confirm correct operation on PVMicro.exe?

A: Any 3D content that does not display in the PVMicro.exe viewer will not display on a phone.

Q: What are UV settings?

A: Content that runs on Mascot Capsule Engine must have texture-mapped UV coordinates for all polygon vertices.

Q: How can I resolve matters when there are polygons I want to have mapped, and other polygons I do not want mapped?

A: This arrangement is not allowed. All polygons are required to, and presumed to have texture mapping. This is a limiting requirement.

Q: The micro converter did not indicate any problems, but the viewer shows no images while the action frame count advances. How can the converted files be displayed in PVMicro?

A: Since the action frames are being played back, the problem is probably related to scaling.

Q: The white texture assigned to the body and hands of my model displays unsuccessfully in the PVMicro.exe display. These parts turn black. What is the problem?

A: Palette index 0 (zero) for the Micro 3D spec is indiscriminately converted to black. Whether translucent or opaque, palette index 0 must be designated to pure black (0, 0, 0).